TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Oleg Kolesnikov

Processing Academic Article Information for Predicting the Quality of Academic Journals

Master of Science Thesis

Examiner: prof. Tapio Elomaa
Examiner and topic approved on 07.11.2018

ABSTRACT

FIRSTNAME LASTNAME: Oleg Kolesnikov
Tampere University of technology
Master of Science Thesis, 44 pages
November 2018
Master's Degree Program in Information Technology
Major: Software Engineering
Examiner: Professor Tapio Elomaa

Keywords: DBLP, JUFO, data collection, data mining

The main method of corresponding scientific ideas and results is to publish articles. The number of scientific journals has been growing rapidly in recent years. All journals are not of the same quality and standard. There are even predatory journals that aim at earning through publication fees paid by the authors. All of this has made it necessary to rank scientific publication forums according to their quality and selectiveness. There is also a Finnish ranking system known as JUFO (julkaisufoorumi).

On the other hand, there are several open electronic libraries available that record information of published articles. Well known examples of such libraries include Google scholar, Semantic scholar, and Digital Bibliography and Library Project (DBLP). The last library is concentrated particularly on computer science publications. Also scientific publishers maintain their own repositories of articles published in their books, journals, and collections. Well known examples include Springer and Elsevier as well as more computing related Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE).

Full article texts of publications that are proprietary to commercial publishers are not usually openly available. Electronic libraries gather all basic information like author names, article title, publication year, and so forth. Sometimes also the abstract of the paper is publicly available. In particular, publishing houses are willing to divulge this information.

It is a natural idea to connect electronic libraries and publication ranking sites together and try to learn to rank journals automatically. This is the setting of this work. This work aims at implementing a program that is able to collect a comprehensive data set consisting of article information and the ranking given for the journal. We concentrate on the DBLP library and JUFO ranking. As a second contribution we validate the feasibility of the proposed approach by applying a couple of machine learning algorithms from the WEKA collection to the data set collected. The experiments show that quite high prediction accuracies can be achieved by using the information gathered from the abstracts of the articles.

CONTENTS

List of Symbols and abbreviations

ACM	Association for Computing Machinery
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
DBLP	Digital Bibliography and Library Project
DLL	Dynamic Link Library
EF	Entity Framework
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
JUFO	Finnish Publication Forum (Julkaisufoorumi)
MVC	Model-View-Controller
ORM	Object-relational mapper
WEKA	Waikato Environment for Knowledge Analysis
XML	Extensible Markup Language

1. INTRODUCTION

Areas of using data mining, machine learning and other technologies related to big data have constantly grown. Widespread use of big data to address different challenges starting from customer needs and up to extremely complicated systems like car traffic management creates a demand for new studies.

The primary goal of this project is to collect a dataset to be used in different kinds of studies. The dataset is based on the metadata contained in the XML dumb file of the Digital Bibliography and Library Project (DBLP). The dump file structure is rather primitive and presents only basic metadata on academic articles. Among others, the metadata includes article title, list of authors, and year of publishing. Although, those attributes are necessary, they are not sufficient. Sources to extract additional data for academic articles should be found. There are numerous APIs to provide article abstract and full texts in the Internet. These APIs are potential sources to add extra data for the articles from the DBLP file.

Another objective is to conduct a set of basic experiments with this data to prove that the collected database is a valid dataset to conduct experiments with. For that purpose, on-shelf machine learning algorithms appear to be a perfect option. On-shelf solutions are tested. On the other hand developing own solution would require a lot of resources. The major idea of experiments revolves around prediction article publishers' ratings according to Finnish Publication Forum (JUFO). Thus, in order to conduct experiments, one must map JUFO rating of publishers onto articles from the DBLP dump file.

Although the research may reveal some interesting insides in the area of text analysis, the major value of the project is that the data collected set may be used in other experiments as well.

Section 2 describes the foundations of the project, namely research questions and input data like dump files with academic articles and journal ratings.

Section 3 concentrates on the technical implementation of the solution. It includes detailed descriptions of serialization the DBLP dump file and storing its content in the database. This section also explains methods of communication with side APIs to extract article abstracts from. Finally, this section outlines challenges connected to assigning ratings to academic articles.

Experiments with the aggregated data are outlined in Section 4. This section includes data preparation as well as running machine learning algorithms on this data. Obtained results are briefly analyzed.

Section 5 contains major aspects of support and maintenance of the solution as well as suggestions on its further development.

Finally, conclusions and their brief analysis are presented in Section 6.

2. BACKGROUND AND RESEARCH QUESTIONS

This section details the foundations of the research. First we introduce the research questions targeted in our work in Subsection 2.1. The next Subsection 2.2 discusses DBLP, which we use as the primary source of scientific article information. It is an example of an electronic library mainly concentrating on computer science articles. Finally, in Subsection 2.3 outlines main features of JUFO publication forum which is a national example of a publication quality ranking site.

2.1 Research Questions

There are several questions this thesis is expected to answer.

The objective of this work is to process article information for certain kind of predictions. Therefore, one of the major questions is what is the sufficient amount of data to make predictions with an appropriate accuracy level? Is it enough to operate only with article titles or additional information (abstract, full text, authors list) is needed? If for more accurate predictions one should use a richer content than just an article title, which additional data exactly should be added given limited technical and financial resources (data base, additional data costs, development costs, etc.) of current project?

There is another critical point one should estimate before starting. Is it possible at all to collect and maintain large amount of data obtained from different sources with no control on them? Mostly, data is received from different side APIs which are quite likely to change over time. Those changes may connect to the response structure, access instruction, and the like.

And last but not least, is it possible in principle to predict the quality of a text based on its word bag? Applying common logical approach, one may assume that there are more or less the same term sets in academic articles irrespectively of the journal rating. In other words, a journal containing a particular set of terms cannot be compared with another journal containing different set of terms. At least, the comparison on term bag seems to be incorrect given that both journals belong to academic sphere. On the other hand, there still may be a possibility of correlation between journal terms and ratings. Depending on the results of studies there may be interesting insides in the area.

2.2 Digital Bibliography and Library Project

Digital Bibliography and Library Project (DBLP) is a project to provide bibliographic information on sources associated with computer science academic data. Those sources are primarily computer science journals [1], thematic conferences, books, and the like. The number of publications provided by the DBLP dump file has constantly grown since 1996 and exceeds 4 000 000 items in total. When it comes to journal articles, the file contains about 1 700 000 journal articles descriptions as for the beginning of 2018 [2].

The DBLP project provides for download its XML dump file with publications. The file is constantly updated [3]. Increasingly, a considerable number of new academic source items appear in the DBLP dump file (which also suggests a constant update of the current project's database). In Figure 1, there is an example of publication item from the XML file.

```
1.  <article mdate="2014-09-05" key="journals/arscom/BelbachirB14">
2.      <author>Hacegravene Belbachir</author>
3.      <author>Imad Eddine Bousbaa</author>
4.      <title>Combinatorial identities for the r-Lah numbers.</title>
5.      <pages>453-458</pages>
6.      <year>2014</year>
7.      <volume>115</volume>
8.      <journal>Ars Comb.</journal>
9.      <url>db/journals/arscom/arscom115.html#BelbachirB14</url>
10. </article>
```

Figure 1. *Publication XML item*

As one may notice from Figure1, an item's structure is rather simple and besides article title, the only valuable extra data for most of the potential data mining experiments is the author list. This is why it is extremely important to supplement existing data with extra attributes for making rating predictions of the DBLP articles more reliable in particular and experiments with data more accurate in general. Obviously, there are two main candidates to serve as extra data, namely article abstracts and article texts. A number of factors suggest the former option. First, extremely few academic staff sources at public disposal provide articles texts without charge whereas there are about a dozen side APIs to provide article abstracts used in current project. Another powerful argument in favor of abstracts is the storage capabilities available for current project. An average article text is 20 times as long as its abstract. The computational and storage resources of current project are limited to store and operate with such a huge dataset.

In addition, the size of an average article abstract contains a sufficient amount of essential information. It normally reflects major ideas of the article. Thus, for the above mentioned reasons, the most preferable option between the two proves to be article abstract.

2.3 Publication Forum

JUFO is a Finnish system of classification of academic article channels. The main aim of the project is to provide a support in the quality assessment of academic research [14]. The number of academic article journals has been growing increasingly during several last decades. The quality and standard of journals may vary considerably. There are even "fake" journals that aim at earning through publication fees paid by the authors. These factors create a need for ranking scientific publication forums according to their quality and selectiveness. For that reason, the Publication Forum has its own set of publisher ratings.

There are 4 different ratings the JUFO project uses for rating evaluation. 0 (zero) means there is no rating evaluation data for the publisher or does not reach 1. 1-3 reflects ratings for publishers in ascending order. For example, a publisher with the rating 3 is considered by be more reliable than that with the rating 2. There is also a native rating classification at the Publication Forum site. 1 refers to basic, 2 refers to leading, and 3 to top.

Besides numerous benefits for researchers including evaluation the average quality of a publisher, suggestions of correction publisher ratings and creating own list of journals, the project allows downloading the whole set of academic channel ratings in a form of dump file which is extremely helpful for current project whose objective is to collect a data corpus and predict JUFO ratings of the articles in this data corpus.

Among others, the JUFO dump file contains the following relevant for this project data: publisher name, ratings for the previous several years, and country of origin.

3. SOLUTION DETAILS

Section 3 concentrates on the implementation of the solution. Subsection 3.1 outlines technical details and tools used in the solution as well as justifies the choices made. The following Section 3.2 describes methods of serialization of the DBLP XML dump file to store its content in the database. The Subsection 3.3 shows technical details of storing JUFO rating dump file in the database. The largest part, Subsection 3.4, explains principles and technical details of extraction of abstracts for the articles from the DBLP dump file. Finally, Subsection 3.5 deals with the challenges associated with assigning JUFO ratings to the articles from the DBLP dump file.

3.1 Common Technical Characteristics of the Solution

3.1.1 Framework, Project Type

Framework .NET is a set of instruments and libraries to provide services for running applications of different type [13]. There is a set of requirements to the development framework in this project. Primarily, those are a reliable and easy-to-work-with database access tool, a rich library to for XML and JSON serialization/deserialization, preferably a high-level communication means over HTTP. Framework .NET meets all above mentioned requirements [13].

Moreover, .NET solutions are built within so called application domains. Application domain is a model that scopes the code execution and resources. In other words, application domain allows uniting different types of projects, like dynamically linked libraries, services, web projects, into a single solution guaranteeing wholeness and safety of code execution.

As the selection of the platform was done, there is a question about the type of project to use for the solution. ASP.NET MVC (model-view-controller) appears to be the most preferable option for a relatively large web project being developed on the .NET platform.

Starting with short theoretical introduction, MVC is a design pattern splitting an application into at least three parts: models, views and controllers [4] and by this provides significant advantages, including loose coupling, extensibility, and cheaper maintenance. The parts MVC pattern consists of are outlined below in the context of the ASP.NET MVC implementation.

ASP.NET MVC fully complies with the MVC architectural pattern and besides it implements numerous features to make development faster and more effective in terms of future code support and maintenance. For example, there is no need to concern about the interaction between controllers and views, ASP.NET provides wide variety of tools to ensure safe and effective interaction.

Models in the MVC architecture represent data used in the application. Particularly, ASP.NET MVC mostly operates with classes, structures, anonymous types and primitive types (strings, integers, doubles, etc.). Generally, in the MVC architecture pattern, model is data to be processed and displayed in an appropriate form for the user. In particular, there are multiple options to process data from the model and pass to the view in the ASP.NET MVC technology. For example, it is possible to display prepared data with the built-in view engine named Razor. For this purpose, it is more natural to pass a model as a native C# type, like an instance of a class or a primitive type. Another quite frequent option is processing the data into JSON format for further parsing with the means of a frontend framework.

Views are responsible for rendering data as graphical user interface (GUI). There are two the most commonly used methods for rendering user interfaces in ASP.NET MVC projects. Those are server-side rendering and client-side rendering. In the first case, rendered layout is passed to the client. The client, or in other words browser, simply displays the layout as it is. This approach is the perfect option for fully static views. However, it appears to be quite a poor choice when it comes to more or less interactive GUI, where the user is allowed to make different kinds of manipulations like selections, changing data, etc. For such occasions, it is worth considering client-side view rendering which is typically made with the means of frontend reactive frameworks or with the asynchronous JavaScript and XML technology (AJAX). Client-side rendering makes it easier to modify the document object model of web pages. Current project utilizes both methods. Server-side rendering is used for pages displaying static data, like search results. Client-side rendering is a preferable tool for document object model modification. For instance, journal names mapper utilizes AJAX requests for updating the layout.

Controllers play a critical role in the MVC architecture pattern. They serve as mediators between models and views making the whole architecture more flexible, adaptable, and extensible. In the ASP.NET MVC technology, the above mentioned features are achieved by implementing a rich collection of data processing methods. This collection also includes means of preprocessing and post processing which is especially useful with respect to universalism. This collection includes filters, binders, handlers, and other mechanisms.

Controllers process data received typically either from incoming web requests or the application database to process it on the data model. Once a result is produced, the controller chooses a view to pass the result. And again, as one may have already used to,

ASP.NET MVC controllers provide lots of options to transfer the data. This is possible due to the complex structure of controllers. Every controller consists of actions, or special methods, web requests address to. Also, inside those actions may be method invocations to retrieve data from the data base. Further on, the output may be passed directly to the view in some of the forms described above or to another action for additional processing.

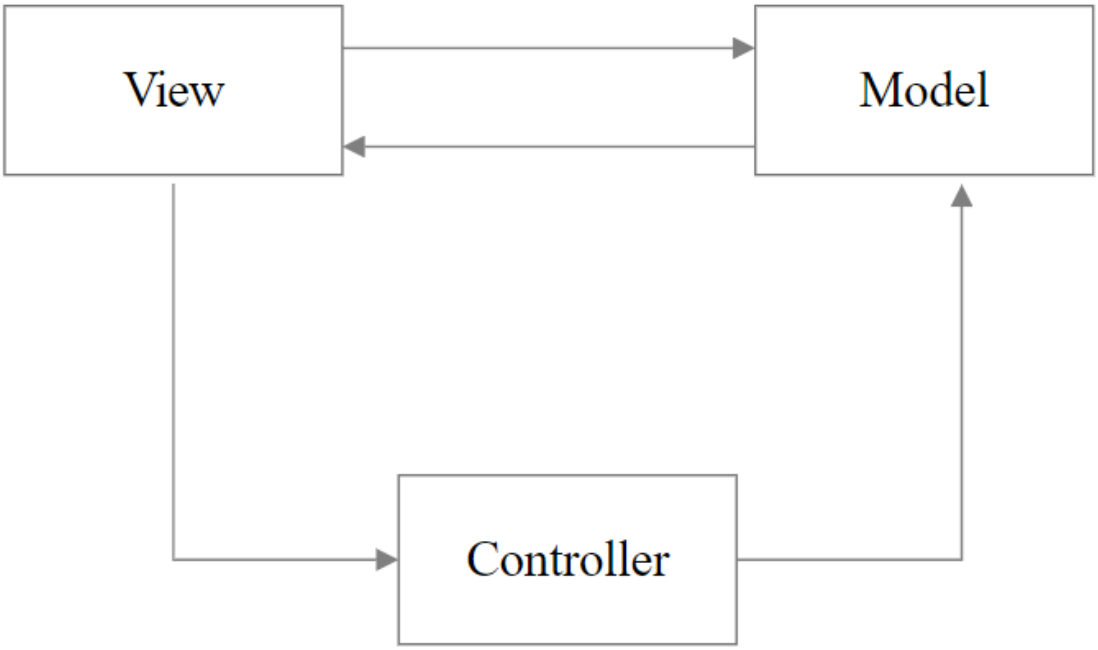

Figure 2. *Simplified MVC architecture base diagram*

Figure 2 shows a core scheme of the MVC architecture. This is an easily expendable architecture pattern; arbitrary number of layers may be added to an existing solution preserving its current architecture. Concerning current solution, there is one additional level to encapsulate business logic. This layer is responsible for exchanging data with the APIs presented in the solution, communication with data base via an object-relation mapper, etc.

3.1.2 Database Tools, HTTP Request-response Processing Approaches

One of the fundamental stones to build an application upon is choosing database access technique. Increasingly, applications have been using object-relation mappers (ORM) as it appears to be an optimal choice and have several important advantages [5]. First, ORMs provide mechanisms for automated casting object-to-table and table-to-object operations, which significantly simplifies development. Another benefit to reduce development time is less volume of coding since there is no need to code on the database

tier. Last but not least, ORM caches data on the application side which results in less number of queries against the database and hence better application performance.

ADO.NET Entity Framework was chosen as an object-relational mapper for a number of reasons. First, to avoid impedance, it is always preferable to apply technologies recommended for the chosen development framework. In case of .NET, the recommended data access technology is Entity Framework (EF) [6]. Also, taking a deeper look into this data access technology reveals practical advantages of utilizing it. Besides all above mentioned common features of ORMs, EF offers different approaches to creating conceptual models: database first, model first, and code first. The developer is able to choose the most favorable way depending on initial conditions. This particular solution utilizes the code first approach. Among many others, LINQ-to-Entities technique holds a special place. It makes it much easier to write queries to extract the data required [8].

However, there are always costs. It is fair to mention that using EF has its own performance considerations. More detailed analysis is presented at the official Microsoft web site [7]. The comparative analysis presented in the article suggests that in the majority of cases performance considerations are less important than then simplicity and reliability EF provides.

The programming solution for this thesis assumes constant communication with side services. All of them either require or support communication over the HTTP protocol. This is why there is a strong need for a robust solution for the purpose.

Among number of tools providing safe and stable communication over HTTP protocol for .NET programs, WebClient [9] is an on-shelf solution from .NET. It appears to be an optimal choice for HTTP request-response processing. Unlike some other tools, like HttpWebRequest or HttpClient, WebClient has a clear set of methods to communicate over HTTP protocol [29]. On the other hand, it has no control over communication on low level and slightly slower compared to others. Since neither speed nor complex control on low level is required for the APIs used in current project, the main feature for HTTP communication tools becomes its simplicity. Therefore, WebClient was selected since it provides plane and easy-to-use methods for upload (download) data from side services.

3.2 DBLP Dump XML File Deserialization

The DBLP dump file is available in XML format only. Hence, a deserialization is needed to store the data from the file in an appropriate format in the database. There are several challenges coming with deserialization of the DBLP dump file.

3.2.1 XML File Formatting

First, the dump file is not guaranteed to be correctly formatted, i.e., multiple nodes may appear on the same line. There is also a possibility of badly structured XML. Some nodes may be missing closing tags, etc. For those reasons, XML formatting and validation is critical.

```
1.  <article mdate="2005-05-27" key="journals/arscom/HellwigV04">
2.      <author>Angelika Hellwig</author> <author>Lutz Volkmann</author>
3.      <title>Maximally local-edge-connected graphs and digraphs.</title>
4.      ...
5.  </article> <article mdate="2017-04-10" key="journals/arscom/CorcinoCG17">
6.      <author>Roberto B. Corcino</author>
7.      <author>Richell O. Celeste</author>
8.      <author>Ken Joffaniel M. Gonzales</author>
9.      <title>Rook Theoretic Proofs Of Some Identities Related... </title>
10.     <pages>11-26</pages>
11.     ...
12. </article>
```

Figure 3. *Example of unformatted XML nodes*

Figure 3 displays a fragment of the DBLP XML dump file with broken structure. At least, two approaches may be applied to parse XML nodes from a poorly formatted file. First approach assumes creating a complex XML parser with sophisticated regular expressions and other tricks to parse a string of arbitrary structure. The major advantage of this option is automatization meaning that the only need for the user or developer is to feed the data to the parser. After that, the program is responsible for file formatting and other preparatory steps. However, there are always costs. A complex utility with many functions inside is always a matter of concern. Thus, this variant allows avoiding the stage of manual formatting but there are development and maintenance costs associated with a complex XML parser written.

An alternative approach is an additional layer, namely, preparatory formatting with a special formatter. This option adds an additional stage and makes the process of the dump file deserialization less automatic but saves sufficient amount of time and it is more reliable. The latter option was selected. However, there is always a possibility to switch to another option later on.

For that project Liquid Studio 2017, which is available for download at the official site of Liquid Studio [10], was selected as an XML formatter. It proved to be fast and reliable. The Liquid Studio is able to format extremely huge files (the size DBLP dump file is exceeds 3 GB as for the beginning of 2018). Still, it should be mentioned that a free license for this tool is available only for a trial period.

```
1.  <article mdate="2005-05-27" key="journals/arscom/HellwigV04">
2.          ...
3.      <title>Maximally local-edge-connected graphs and digraphs.</title>
4.      <year>2004</year>
5.      <volume>72</volume>
6.      <journal>Ars Comb.</journal>
7.      <url>db/journals/arscom/arscom72.html#HellwigV04</url>
8.  </article>
9.  <article mdate="2017-04-10" key="journals/arscom/CorcinoCG17">
10.     <author>Roberto B. Corcino</author>
11.     <author>Richell O. Celeste</author>
12.     <author>Ken Joffaniel M. Gonzales</author>
13.     <ti-
    tle>Rook Theoretic Proofs Of Some Identities Related To Spivey's Bell Number F
    ormula.</title>
14.         ...
15. </article>
```

Figure 4. *Formatted XML fragment*

Figure 4 displays the same fragment of the DBLP dump file as Figure 3 after formatting. Formatting the dump file proved to be essential. The XML parser spotted only just over 1 000 000 articles before formatting whereas the number of articles spotted after it exceeds 1 400 000. Thus, the formation gave a raise of articles by approximately 40%.

3.2.2 XML File Deserialization and Saving It into Database

The XML parser reads the preliminarily formatted XML file line by line. At this stage, every line corresponds to an XML node opening tag. Once the parser receives a desired node, it casts this node from the file into a .Net XML type (XElement). Further on, the XML object created is deserialized into a model associated with Entity Framework and therefore database. Next, Entity Framework saves the obtained XML data into the database.

The DBLP dump file is not bounded only by articles. It contains items from several academic output types including conference papers, books, and electronic academic sources.

The solution, as one can see from the code snippet below (Program 1), may be expanded to collect more types of data simply by adding new cases like proceedings, books, or web resources into the *switch* operator. Therefore, no design changes are needed in case new types of data have to be extracted from the DBLP dump file.

```
using (XmlReader reader = XmlReader.Create(sourcePath))
{
    reader.MoveToContent();
    //read from file node by node
    while (reader.Read())
    {
        if (reader.NodeType == XmlNodeType.Element)
        {
            try
            {
                XElement xe = null;
                BaseItem item = null;

                switch (reader.Name)
                {
                    //search article nodes
                    case "article":
                        xe  = (XElement)XNode.ReadFrom(reader);
                        item = new Article(xe);

                        using (var context = new ArticleContext())
                        {
                            //check if the article is not in db
                            if (!context.Articles
                                    .Any(a => a.Title == item.Title))
                            {
                                //add article
                                context.Articles.Add((Article)item);
                                context.SaveChanges();
                            }
                        }

                        break;

                    default:
                        break;
                }
            }
        }
    }
}
```

Program 1. *Parsing XML nodes into a model and further via EF saving them to the database*

3.3 JUFO Dump Rating File Deserialization

The JUFO rating file is provided in an Excel-readable format (.csv). The document contains numerous columns including publisher title, publisher type, and publisher rating for the last several years and others attributes. The dump rating file is available at https://www.tsv.fi/julkaisufoorumi/haku.php?lang=en. The dump file should be casted into a usable format, preferably into a SQL data table.

There is a commonly acknowledged approach among software developers. It says, that one should always stick to the most simple and effective solutions. In case of casting data from Excel to the database, Microsoft SQL Server, default database server for .Net solutions, provides a wide variety of tools for transferring data between Excel documents and SQL tables. The SQL Server Import and Export Wizard is one of them [11].

With the means of this tool, a table containing publisher names, countries of origin and rating was generated for JUFO rating dump file. Table name in the solution database is JufoRatings.

3.4 Abstract Seeking

As all articles and rating are stored in the database, it is time to take next step. There is not much use in solely article titles especially when it comes to data mining and different kinds of statistical predictions. In Section 2, it was decided to complement academic articles with abstracts. For that reason, abstract seeking and further extraction is a crucial task related to the articles stored in the database.

The process of filling the database with article abstracts consists of two major parts: finding abstract and storing it to the database. This two-level approach conforms to the so called SOLID principles of object-oriented software development like separation of concerns and loose coupling [26]. Program 2 gives a description of the "upper" part of the combination.

```
1.  /// <summary>
2.  /// uploads abstracts
3.  /// </summary>
4.  public static void  UploadAbstracts(int itemsToSkip, int itemsToTake)
5.  {
6.      //get articles that have no abstract
7.      var articlesWithoutAbstract = GetArticlesWithoutAbstract(
8.                                                  itemsToSkip, itemsToTake);
9.      var log = new XElement("Record");
10.
11.     if (!articlesWithoutAbstract.HasValue())
12.         log.Add("No articles came");
13.
14.     foreach (var article in articlesWithoutAbstract)
15.     {
16.         log.Add(new XAttribute("ArticleId", article.Id));
17.
18.         var abstractSeeker = new AbstractSeeker().TryToFindAbstract(article);
19.
20.         log.Add(new XAttribute("HasAbstract", abstractSeeker.HasValue()));
21.
22.         //abstract not found
23.         if (!abstractSeeker.HasValue())
24.             continue;
25.
26.         //create abstract object
27.         var articleAbstract = new Abstract();
28.         articleAbstract.Id = article.Id;
29.         articleAbstract.Text = abstractSeeker;
30.
31.         SaveAbstract(log, articleAbstract);
32.
33.         //add record to log
34.         ...
35.     }
36. }
```

Program 2. *Seeking abstract for articles*

As it was mentioned, principles SOLID of object-oriented programming allow writing code relying on abstractions rather than implementations. At this stage, as one can see, the actual implementation of abstract seeking method - *TryToFindAbstract* - is not important. If the logic of abstract seeking needs to be modified, one should only make changes to the particular method while the core logic remains unmodified. This is an example of a stable construction that allows extending the application.

A number of digital libraries with academic article abstracts are available on the Internet. No source necessarily contains all the needed abstracts, hence, in the worst case the abstract for an article must be sought across several sources.

Abstract extractions from those sources are tasks of different complexity. Some sources provide abstracts within a response with a rather simple structure. Other sources, besides special requirements to the HTTP requests, may contain several variants of abstracts for articles with similar titles. Therefore, the most reasonable approach appears to be prioritize sources according to the amount of resources required to develop an abstract extraction method and the volume of useful data that can be extracted from the source.

Below, in the following subparagraphs, a brief description of every API involved is given. The APIs are arranged in order of their priority.

3.4.1 Semantic Scholar API

Semantic Scholar is a project of Allen Institute for Artificial Intelligence. It is an open academic search engine [16]. This is a complicated project with lots of different features yet what is the most valuable for the current project is the Semantic Scholar API.

The Semantic Scholar API is an open API used for abstract extraction at the DBLP web site. The API is the absolute leader among digital libraries to provide abstracts presented here. Whereas the development costs for the client method to communicate with this API are comparable with those of communication functions for other APIs, the rate of abstracts returned is roughly 52% (760 000 abstracts for 1 460 000 articles as for 09.06.2018). Thus, statistically, this source provides abstracts for over half of all articles stored.

A typical response returned by the API is in JSON format and contains a set of paper metadata i.e. article title, authors, abstract and others. There is not necessarily desired article metadata among the items returned. For that reason, some filtering is conducted by the current project application.

The API requires no key.

3.4.2 Springer API

Springer has a huge web resource to provide data from about 3 000 scientific journals and 250 000 books to name a few. It has an advanced data corpus available (under different conditions) for students, researchers, and everyone interested in science [15].

Springer has developed multiple APIs for those interested to access their freely available content. The use of the content must be noncommercial. The API documentation may found here: https://dev.springernature.com (as for 20 June 2018).

Similarly to the Semantic Scholar API, a Springer API response normally consists of a set of article metadata with all the essential fields: article title, abstract, author, venue.

However, there are two important differences to mention. Unlike the Semantic Scholar API, the Springer API requires a key to be sent in the request string. Moreover, the number of API calls with one API key is restricted by 5000 hits per day.

Fortunately, the Springer API allows registering and using a bunch of keys for the same user. Thus, theoretically, one should be able to send as many requests as needed.

Springer API keys registered (as for 25.06.2018):
4fcbbf629991f8919b846404ad651387,
ac06264c3272582e55cc8a71dac0926f,
05ddc6784584b41171d09a665a60b8e1,
23a985da537de7b85916ccdc766dcc0d,
0b1f7bb22775fe09d8eec6777c3fb327, and
c7dc2d2a2c4a03be81c552535f8c4e2e.

Since all methods to extract abstract for the first ranked three APIs are structurally analogous, the snipped is presented only for one abstract extraction method. The flow includes the following steps: input validation, request string formation, getting response from the API, deserializing it, and filtering only desired abstract if exists at all. Program 3 displays a typical method of getting and deserialization abstract.

```
1.  /// <summary>
2.  /// get abstract from springer api
3.  /// </summary>
4.  /// <param name="article"></param>
5.  /// <returns></returns>
6.  private string GetAbstractFromSpringerApi(Article article)
7.  {
8.      //not to search on particular meaningless titles
9.      if (ArticleTitlesNotToSearch.Any(a => a == article.Title))
10.         return null;
11.
12.     var articleTitle = article.Title.Trim('.');
13.
14.     //prepares request ull
15.     var fullUrl = PrepareSpringerRequest(articleTitle);
16.
17.     //get response from api
18.     var response = SendRequest(fullUrl);
19.
20.     //deserialized response
21.     var apiDeserializedResponse =
22. GetAbstractFromApiJsonResponse<SpringerApiResponse>(apiResponse: response);
23.
24.     if (apiDeserializedResponse == null)
25.         return null;
26.
27.     ///get abstract
28.     var articleAbstract = apiDeserializedResponse
29.       .Records?.Find(p => p.Title == articleTitle)?.Abstract;
30.
31.     return articleAbstract;
32. }
```

Program 3. *Typical method to get a deserialized abstract to save it into database*

3.4.3 IEEE Xplore Api

IEEE is a professional technical organization to promote technology and technological innovations [17]. It is a huge organization that holds numerous annual conferences as well as publishes dozens of scientific journals. One of this organization's projects is the IEEE Xplore Digital library offering millions of scientific publications. The Library also has an API to extract various metadata including academic article abstracts.

Among the ones presented here, this API is ranked third for two major drawbacks. First, the IEEE Xplore Digital library apparently contains academic articles and conference items only from the publication channels associated with IEEE project. Whereas it is extremely helpful in some occasions, it proves be of no use to conduct search for any source not connected to the IEEE project. Also, the maximum number of calls allowed per day is 200. This restriction is another reason for poor ranking.

IEEE Xplore API keys registered (as for 25.06.2018): 8ewaer7s7gq98rqff7kyuugh.

Additionally, it appears to be reasonable to take these two points into account. In order save few enabled API calls per day as well as not to send ineffectual requests, the following restriction was added:

```
1.  //IEEEXplore contains metadata only on their own venues
2.  if (!article.Publisher.Contains("IEEE"))
3.        return null;
```

Program 4. *This plain logic restricts calls to IEEE Xplore API. Names of all publication channels associated with IEEE contain this abbreviation*

This primitive restriction (shown in Program 4) enables calling the IEEE Xplore API only if the article publisher is associated with the IEEE project.

3.4.4 Common Parser for APIs Returning Responses in JSON

Since all above described APIs return data in JSON format, it is reasonable to parse answers with a single deserializer.

C# specifies generic methods for these kinds of purposes. This is a very powerful feature, allowing creating methods that operate with open-ended types converting them into closed (definite) types at run time [27]. Thus, the generic parser (Program 5) accepts string of data in JSON format and safely deserialized it to the open-ended type that closes only at run time.

```
1.  /// <summary>
2.  /// generic method to deserialize api responses in JSON format
3.  /// </summary>
4.  /// <param name="apiResponse">may contain a number of abstracts</param>
5.  /// <returns></returns>
6.  private T GetAbstractFromApiResponse<T>(string apiResponse)
7.          where T : class, new()
8.  {
9.      if (!apiResponse.HasValue())
10.         return null;
11.
12.     var resp = new T();
13.     try
14.     {
15.         //deserialize json response
16.         resp = JsonConvert.DeserializeObject<T>(apiResponse);
17.     }
18.     catch
19.     {
20.         //log error
21.         ...
22.         return null;
23.     }
24.
25.     return resp;
26. }
```

Program 5. *Generic parser for any API response in JSON format*

3.4.5 Scopus Abstract Retrieval API

The Scopus API is created and maintained by the Elsevier project [25]. This is a project to assist researchers and professionals around the globe for the benefit of humanity, as states Elsevier's motto.

The Elsevier has a huge database. It provides 1.4 billion cited references which make it potentially one of the major sources to maintain the database of current project.

Web method to extract abstract from this API is to be developed.

Key : 571f0f1376d2a8e18882e268f2e000d2

3.4.6 ACM Bulk File

The Association for Computing Machinery is the world's largest scientific society in computing academic field with almost 70 years of history [18]. It is the only academic materials resource among the ones presented here not to provide a public API. Instead, ACM permits, under certain conditions, downloading their bulk file with metadata. The ACM bulk does not appear to be effective since it contains metadata mostly on ACM publication channels and proceedings.

However, it may turn out to be highly useful in case of extending current project. No method to extract data from ACM bulk file has been developed. The development of this software remains for possible future stages.

3.4.7 Google Scholar

Despite providing the richest content, Google Scholar does have no an actual API. As a result, the only way to receive Google Scholar's response in a form to be parsed programmatically (in this case, it is HTML), is to write an Http client with a humanlike behavior. That is the major challenge one faces using Google Scholar. This is why this source is ranked the last and should be considered in rare cases of extremely important data that cannot be extracted from other sources.

Moreover, Using Google Scholar assumes a complex method consisting of several parts: source link extraction, defining the source, and applying the corresponding abstract extraction method.

First part is to extract link from the Google Scholar response.

```
/// <summary>
/// takes link by article name among multiple links from google scholar
/// </summary>
/// <param name="articleTitle">
/// <returns></returns>
private string GetLinkForAbstractFromGoogleScholar(string articleTitle)
{
    var result = string.Empty;

    //google scholar response
    var resp = SendRequest(Definitions.GOOGLE_SCHOLAR_URL + articleTitle);

    var linksMatches = Regex.Matches(resp, Definitions.GOOGLE_SCHOLAR_LINK,
                                     RegexOptions.IgnoreCase);

    //nothing found
    if (linksMatches == null || linksMatches.Count == 0)
    {
        log.Add(new XElement("GSResult", "No relevant links provided"));
        return result;
    }

    foreach (Match m in linksMatches)
    {
        //link for the article
        if (m.Groups[2].Value.ToUpper() == articleTitle.ToUpper())
        {
            result = m.Groups[1].Value;
            break;
        }
    }

    return result;
}
```

Program 6. *Method to extract link from Google Scholar responses*

The code snippet displayed above in Program 6 sends a request to the Google Scholar and processes the answer by extraction a link for a source containing the sought abstract. The tool most widely used for purposes of text fragment extraction is regular expressions. Below is the regular expression which effectively extracts links from the Google Scholar response structure as for 27.08.2018.

```
/// <summary>
/// regex for google scholar link to the article source
/// </summary>
public const string GOOGLE_SCHOLAR_LINK =
          @"<h3.*?class=""gs_rt"".*?<a\s+href=""(.*?)"".*?>(.*?)<";
```

Program 7. *Regular expression for link extraction*

Obviously, there is a risk of changing the structure of Google Scholar responses. Taking into account the complexity of the response structure, potential costs of support and maintaining this method may be high. This fact also influences the rank Google Scholar receives.

After the link for a source containing the sought article abstract has been extracted, the next step is to send a request to the source. Theoretically, there is an indefinite amount

of digital libraries and, correspondingly, links for them. Thus, an indefinite number of different links may occur in Google Scholar responses. Generally, every source would require its own method for abstract extraction. All above mentioned factors narrow down a possible solution to a dictionary of key-value pairs. Key is a link for a particular and value is a pointer to the method implementing abstract extraction from this source.

Finally, there is last but not least point with regard to using Google Scholar. As Else [28] points out, it is a very complicated, if feasible at all, task to extract data programmatically from Google Scholar.

To achieve this, a humanlike behavior should be simulated. Requests should have varying frequency whereas the number of requests should be relatively small.

In this project only first part of the algorithm, namely link extraction, has been implemented. The second part remains for the future development.

3.5 Rating Assigning from JUFO File to DBLP Publishers

3.5.1 Mapping Journals Names Problem

Since one of the objectives of this thesis is prediction of journal ratings of articles, JUFO ratings should be mapped onto article publishers.

Journal names from the JUFO dump file and their ratings are stored in the separate table named JufoRatings. Correspondingly, publisher names from the DBLP dump file are stored together with articles (table Articles).

Let us call ‘direct mapping’ an operation when the rating of a journal from the JUFO dump file is assigned to the journal from the DBLP dump file that has exactly the same name. Thus, direct mapping produced only 30% of articles with JUFO rating. In the words, only 30% of records in the table Articles had a value for the attribute JufoRating.

Partly, a minor reason of such a poor result is that the JUFO dump file misses some of the publishers from the DBLP dump file. However, the main reason for this is specific spelling of the majority of journal names in the DBLP dump file.

For example, DBLP data has a significant amount of journal names containing reductions of words, or simply abbreviateions. There are several examples:

```
1. IEEE Trans. Instr. Measurment = IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASU
   REMENT
2. J. Intellig. Transport. Systems = JOURNAL OF INTELLIGENT TRANSPORTA-
   TION SYSTEMS
3. J. Visualization = JOURNAL OF VISUALIZATION
```

Figure 5. *Publisher names spelling differences. The left part is publisher name from DBLP dump file, the right part is corresponding name form JUFO ratings dump file*

Figure 5 shows examples of spelling differences of publisher names. This situation tangles mapping of publishers and ratings. Therefore, a decision should be taken on this issue. There are several approaches to solve publisher names mapping problem: manual, semi-automatic, and automatic.

Generally, manual changes are quite a poor option in software development. The very idea of software assumes automatic processes in accordance to given instructions i.e. algorithms. Besides that it would require a huge amount of time to find all publisher names from the DBLP dump file to map and figure out their corresponding names from the JUFO dump file.

Another alternative, a fully automatic solution proves to be the most preferable in majority of cases. However, taking into consideration relatively small amount of publishers to map (just over 1700 as for 09.06.2018) and resources required to implement this kind of solution, a fully automatic solution does not approve to be the best option. For instance a solution outlined by Yang Hua and others [12] assumes implementation of a complicated statistical algorithm for a combination of complex rules. It would require an enormous effort to develop a program of this complexity. Also, there no 100% guarantee of correct matching since there is potentially indefinite number of rules to be applied for mapping. One cannot exclude the case when several rules are applicable for mapping.

Therefore, an optimal approach appears to be a “golden mean” involving comparatively little coding and an easy way for the user to map publisher names from a set of selected options. Thus, a semi-automatic approach was chosen.

3.5.2 Semi-automatic Solution for Mapping

The task of publisher name mapping does not have clear criteria to map names with: there are random word reductions, abbreviations as well as omitting preposition and other cases that can occur in the DBLP file. Thus, it is logical to execute the mapping iteration by iteration replacing or adding new rules to the existing ones until there are either no unmapped publisher names or no candidates left to map with. For that reason, the mapping algorithm was designed the way to encapsulate mapping instructions into a method (FindMappingOptionsForOnePublisherName). This approach complies with

best practices of object-oriented programing (SOLID) and allows changing mapping instructions while the core logic (semi-automatic) remains untouched.

For every unmapped publisher name in the DBLP dump file, the algorithm provides a set of possible mapping options to select from. It is the user responsibility to pick up the right option among presented.

```
1.  /// <summary>
2.  /// returns list of possible options from jufo rating paper
3.  /// for all alrticle records that are missing jufo rating publisher name
4.  /// </summary>
5.  /// <returns></returns>
6.  public static
7.      List<DblpNameJufoOptionsPair> GetMappingOptionsForAllPublisherNames()
8.  {
9.      var notMappedArticlePublisherNames = new List<string>();
10.
11.     //get all unmapped publisher names
12.     using (var context = new ArticleContext())
13.     {
14.         notMappedArticlePublisherNames = context.Articles
15.             .Where(a => a.PublisherNameInJufo == null ||
16.                         a.PublisherNameInJufo == string.Empty)
17.             .Select(a => a.Publisher)
18.             .Distinct()
19.             .ToList();
20.     }
21.
22.     //nothing came
23.     if (!notMappedArticlePublisherNames.HasValue())
24.         return null;
25.
26.     //possible options for every unmapped publisher name
27.     var optionsForUnmappedPublisherNames = notMappedArticlePublisherNames
28.         .Select(n => new DblpNameJufoOptionsPair(
29.             dblpPublisherName: n,
30.             jufoOptions: FindMappingOptionsForOnePublisherName(n)))
31.         .ToList();
32.
33.     return optionsForUnmappedPublisherNames;
34. }
```

Program 8. *Selecting all possible options for every unmapped publisher name according to the instructions written in the FindMappingOptionsForOnePublisherName*

The method from the sample above (Program 8) produces a list of distinct publishers from the DBLP dump file that has no matching in the JUFO dump file. After that, a set of possible matching candidates is selected for every element form the produced list of unmapped publishers.

Once user selects an option to map a publisher name, the GUI sends an AJAX request to the server that saves the selected mapping option. It is a common practice to use asynchronous requests to the server in order to avoid reloading the whole page. It is especially the case when the page contains a fairly large amount of data to extract from the database. Also, the choice of asynchronous updating the web page affects positively user experience.

The following method is executed to map one distinct publisher from the DBLP dump file.

```
1.  var articles = new List<Article>();
2.
3.  //get all articles with specified publisher name and update them
4.  using (var context = new ArticleContext())
5.  {
6.      articles = context.Articles.
7.          Where(a => a.Publisher == dblpPublisherName)
8.          .ToList();
9.
10.     articles.ForEach(a => a.PublisherNameInJufo = jufoPublisherName);
11.
12.     context.SaveChanges();
13. }
```

Program 9. *Mapping article publisher names with a selected option*

Program 9 selects all items with the specified publisher name and assigns the value of the corresponding JUFO publisher to the PublisherNameInJufo property of every item. This mapping allows assign JUFO ratings to the items via the property PublisherNameInJufo.

Rating assignment is carried out directly, i.e. with the means of a SQL query. If necessary, a simple method to update JUFO ratings from the GUI interface may be developed.

4. EMPIRICAL EXPERIMENTS

This section describes data training with machine learning algorithms. Subsection 4.1 concentrates on the preparation of data to be fed to selected machine learning algorithms. Subsection 4.2 outlines data processing with selected algorithms as well as displays the output. The last Subsection 4.3 briefly discusses the results obtained.

Although the major contribution of this project is collecting a comprehensive set of data to serve as the base for further various studies, some basic experiments with the collected data must be conducted. The project WEKA - Waikato Environment for Knowledge Analysis - perfectly fits this objective [19]. This is one of the most prominent projects when it comes to the data mining field. WEKA provides tools for practically any data mining related kinds of research: classification, clustering regression and the like. Weka provides graphical user interface instead of an API communication via some protocol. This is a very powerful software product. Unfortunately, WEKA has not API or other features to be connected programmatically to side projects.

Since this type of side software is used for current project's experiments, it does not allow automatic integration with the current project. Therefore, empirical experiments are becoming a separate part of the work. Several preparative steps required are described in the following subchapter.

4.1 Preparation of .arff File

Generally, the WEKA Explorer (graphical user interface for WEKA project) accepts a number of data formats including SQL data tables. Still, as practice shows, the data format WEKA Explorer works best with is the arff format. An arff file is easy to feed to the WEKA Explorer and the data from the file is available without any additional operations with it. For above mentioned advantages of arff format it was selected to be fed.

```
@RELATION article-rating

@ATTRIBUTE juforating {1, 2, 3}
@ATTRIBUTE text string

@DATA
2,'Catchments show a wide range of response...
1,'This survey reviews fundamental concepts of...
1,'The purpose of this monograph is to introduce...
1,'This monograph is devoted to random set theory...
1,'The present monograph studies the asymptotic behavior...
1,'This monograph presents an overview of universal...
1,'This tutorial treats the fundamentals of polarization...
1,'Concentration inequalities have been the subject...
1,'This manuscript comprises a tutorial on traditional...
...
```

Figure 6. *Structure of a document in the arff format*

In principle, the arff format is similar to the csv one. This format presents data in the form of rows. Every row consists of a set of data attributes. As one may see from the example above (Figure 6), in this case, rows contain JUFO ratings and article abstracts. An arbitrary number of attributes may be added if there is a need.

Since SQL server does not support direct casting to the arff format, the data should first be transformed into an intermediate format. A perfect candidate for this intermediary is the csv format since it is supported by the SQL server and can easily be casted to the arff format.

Thus, in order to generate a document in the arff format, one needs to convert data from the database into the csv format and further to arff by making minimal changes, like wrapping text in quotation marks, if necessary.

4.2 Experimental Data Processing Result

After feeding 8000 abstracts to the WEKA Explorer and applying two different data mining algorithms, the program produced a result described and analyzed below.

Again, one should mention that current project is focused mainly on data gathering, maintaining the database and increasing the number of academic data sources and, by this, the size of the dataset in the database. Taking into consideration this fact, only basic experiments with the data should suffice to demonstrate the ability of the collected data to serve as a dataset for further more sophisticated experiments. For these grounds, in the data mining experiments conducted in the context of this project, the most primitive rules are applied.

To be more precise about the simplicity of experiments conducted in the context of current project, let us consider an example. It is a widespread practice to use a bag of words for natural language data processing [22]. Depending on the words selected into the bag of words, the experiment may follow different objectives. Bag of words

typically excludes so called “junk words” like articles, prepositions and other word meaningless for most of the experiments in natural language processing. No bag of words is used for this project’s experiments.

Two different algorithms were selected to ensure independence of the results on one algorithm.

One of the selected algorithms among provided by WEKA is the J48. As Aljawarneh [21] points out, J48 is a very helpful algorithm when it comes to building plain and easy to understand models. Also, this algorithm allows building small trees and, therefore, avoiding data overfitting.

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     ArticlesWithRatings-weka.filters.unsupervised.attribute.StringToWordVector
Instances:    8000
Attributes:   1001
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
```

Figure 7. *Initial data for J48 classifier*

As one can see from the Figure 7, 8000 rows were fed to the algorithm. Total number of attributes or different words is just over 1000. Standard data processing with 10-fold cross validation mode has produced the following result. The accuracy of prediction is over 78%.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        6244               78.05   %
Incorrectly Classified Instances      1756               21.95   %
Kappa statistic                          0.6344
Mean absolute error                      0.1616
Root mean squared error                  0.3647
Relative absolute error                 40.3348 %
Root relative squared error             81.4765 %
Total Number of Instances             8000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,772    0,138    0,779      0,772    0,775      0,634    0,823     0,740     1
                 0,817    0,179    0,809      0,817    0,813      0,638    0,818     0,738     2
                 0,673    0,049    0,678      0,673    0,675      0,626    0,823     0,536     3
Weighted Avg.    0,781    0,146    0,780      0,781    0,780      0,635    0,820     0,712

=== Confusion Matrix ===

    a    b    c   <-- classified as
 2388  539  168 |    a = 1
  536 3148  169 |    b = 2
  142  202  708 |    c = 3
```

Figure 8. *Results obtained with J48 classifier*

Figure 8 displays detailed description of results obtained with J48 classifier. For example, confusion matrix shows accuracy of classification for every rating.

Another algorithm selected for this project is the Random Forests algorithm. As Breiman puts it [30], this is a combination of tree predictors. Random Forests technique appears to be an interesting candidate due to the following advantages [24]: generally, Random Forests algorithms produce quite accurate results, and what is more important for our case, is that the can work with a huge amount of attributes where each attribute is weak or does not carry much information. Assuming that we do not use bag of words and the number of attributes is quite large, over 1000, the Random Forests algorithm is likely to produce relatively accurate output.

```
=== Run information ===

Scheme:       weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
Relation:     ArticlesWithRatings-weka.filters.unsupervised.attribute.StringToWordVector-R2-W1000-prune
Instances:    8000
Attributes:   1001
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner
```

Figure 9. *Initial data for Random Forests*

As one can see for Figure 9, conditions applied for Random Forests algorithm are the same as those for the J48 algorithm, namely 10-fold cross validation and exactly the same set of attributes.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        6757               84.4625 %
Incorrectly Classified Instances      1243               15.5375 %
Kappa statistic                          0.7319
Mean absolute error                      0.237
Root mean squared error                  0.3015
Relative absolute error                 59.1453 %
Root relative squared error             67.3657 %
Total Number of Instances             8000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,870    0,091    0,858      0,870    0,864      0,777    0,956     0,939     1
                 0,926    0,180    0,827      0,926    0,873      0,747    0,956     0,953     2
                 0,474    0,007    0,907      0,474    0,623      0,624    0,960     0,823     3
Weighted Avg.    0,845    0,123    0,850      0,845    0,837      0,743    0,957     0,930

=== Confusion Matrix ===

    a    b    c   <-- classified as
 2692  374   29 |    a = 1
  265 3566   22 |    b = 2
  180  373  499 |    c = 3
```

Figure 10. *Results obtained with Random Forests algorithm*

Figure 10 shows that prediction accuracy rate for the Random Forests algorithm is almost 85 percent. Results show that two independently working algorithms produced comparable results which may be an interesting base to conduct new experiments upon.

4.3 Discussion of Obtained Results

The results obtained reveal several interesting topics to discuss.

First, similar outputs of two independently functioning algorithms (the rates of prediction accuracy are 78 and 84 percent respectively) suggest that the data fed to those algorithm is suitable for dealing with for conducting data mining experiments. This means that one of the major objectives of this solution is probably achieved.

Another interesting fact to mention is the rate of prediction accuracy. It reaches as much as almost 85 percent for the Random Forests algorithm. Such a high accuracy rate appears to be very curios. Is there a strong correlation between the set of words used in articles and the rating of their publishers or it is merely a statistical paradox that is possible solely with concrete dataset used for the experiments?

5. FURTHER RESEARCH AND DEVELOPMENT

This section addresses to support and maintenance as well as potential development of the solution. Subsection 5.1 describes major aspects of support and maintenance. Those aspects include adding new articles to the database, assigning ratings and searching abstracts for newly added articles. The next Subsection 5.2 shows the opportunity of using current solution as a side API for other projects. The last Subsection 5.3 suggests development current solution by adding extra data on article authors.

This project is primarily aimed to be used as a base for various studies in data mining, machine learning, and related scientific fields. However, there are a number of points for maintenance and possibilities for further development.

5.1 Extending and Maintaining the Solution

Since the project aims to seek for data from side sources rather than generate it, active maintenance is essential. With regard to maintenance, there are several points to take care of.

First, with every new portion of articles obtained from one source, for instance from DBLP, it is necessary to save only new items, and further map publisher names, JUFO ratings, search abstracts from available APIs, etc.

Second, the APIs currently included in project are likely to transform in the course of time. This is also critically important. The next subchapters describe those aspects of maintaining in more details.

5.1.1 Synchronizing with DBLP Dump File Updates

As an updated XML dump file is released at the DBLP website, one should add new articles to the database. The simplest method to do that appears to find the latest date of adding among those articles in the database and compare it with adding date of every article parsed from the DBLP dump file. If adding date to DBLP of an article is greater than the latest date of adding in the database, it is a fresh article.

```
1.  /// <summary>
2.  /// returns the latest date of adding
3.  /// </summary>
4.  private static DateTime GetLatestDateOfAdding()
5.  {
6.      using (var context = new ArticleContext())
7.      {
8.          try
9.          {
10.             return context.Articles.Select(a => a.DateOfAdding).Max();
11.         }
12.         catch
13.         {
14.             //in order not to clone data in the data base
15.             //logging
16.             return default(DateTime);
17.         }
18.     }
19. }
```

Program 10. *Returns the latest date of adding articles*

Code in Program 10 calculates the adding date for the newest article (in terms of adding this article to the DBLP dump file) in the database.

To sum up the description of the dump file updating, once in a while one needs check for updates the DBLP dump file, copy it to the server and launch updating from the GUI interface.

5.1.2 Adding Ratings and Abstracts to New Articles

After a fresh portion of articles has been saved, it is time to assign ratings to the article publishers as well as search abstracts for them. At the moment, all these three processes are separated. In other words, processes are to be launched consequently from the GUI.

However, the algorithm of mapping can surely be modified to potentially map more publication channels form DBLP and JUFO.

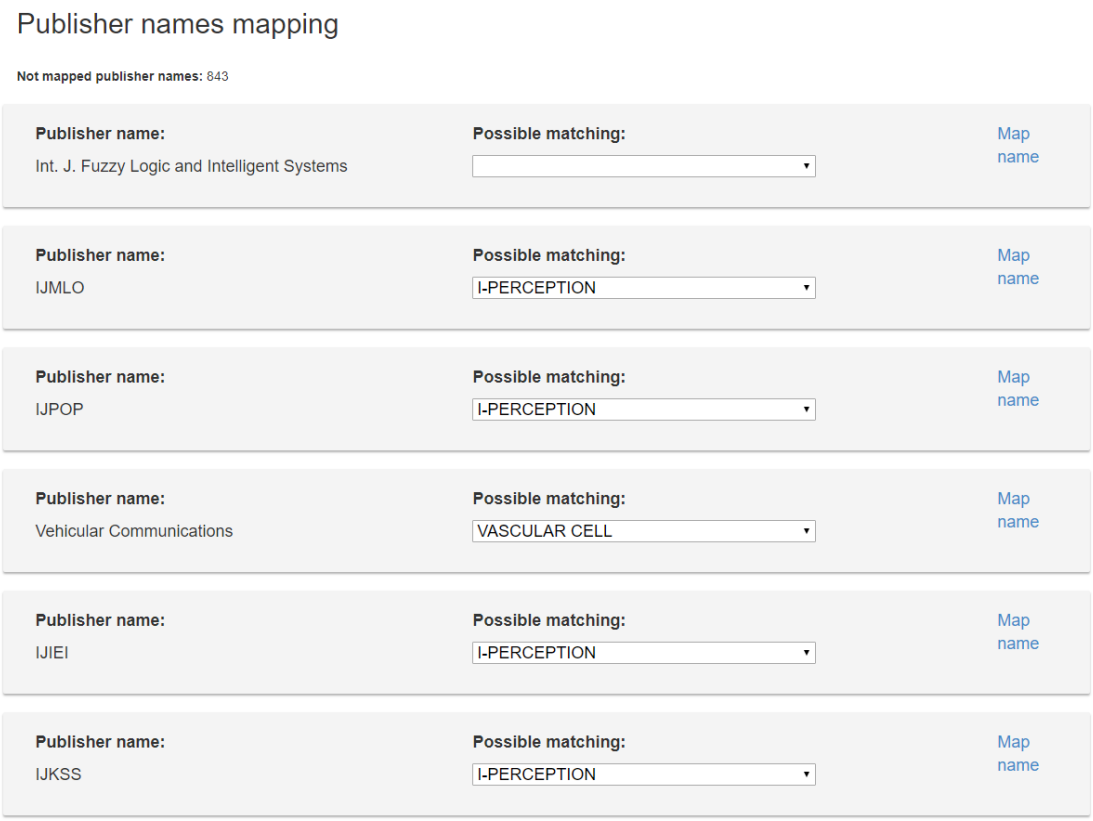

Figure 11. *Publisher name mapper interface*

Thus, speaking about mapping algorithm modifications, there are still several hundreds of unmapped publishers from the DBLP XML file. As one can notice from Figure 11, the majority of unmapped names are abbreviations starting with IJ characters, which with a high probability means *international journal.* At the moment, the core idea of the mapping algorithm is based on acronyms. The mapping algorithms takes first letters of every word in the title (except junk ones like ‘of’, ‘in’, ‘the’, etc.) of DBLP dump file publishers and examines them for matching acronyms from JUFO. Candidates make up selection list.

However, in case a title in DBLP consists only of one word, for instance, ‘IJCST’, it is quite probably that the title is an acronym itself. Unmodified, the mapping algorithm would create an acronym from an acronym, returning only one letter ‘I’ which is statistically not a fully correct approach for such cases. Instead, it appears absolutely logical, to include the following logic to the mapping algorithm:

```
1.  //one word, starts with IJ -
     likely an abbriviation starting with 'International Journal'
2.  if(words.Count() == 1 &&
3.     Char.ToLower(words.First().First()) == 'i' &&
4.     Char.ToLower(words.First().Skip(1).First()) == 'j')
5.       return words.First();
```

Program 11. *Prevents creating an acronym from a title that is highly likely an acronym*

The code displayed in Program 11 has been added to the mapping conditions, but has not been tested thoroughly. It assumes that a one-word title starting with IJ characters is an acronym itself and there is no need to make acronym of it. Testing and modifying remains for further development.

Similarly, the mapping algorithm can be modified to map publishers on some other criterion.

5.1.3 Maintenance Aspects Associated with Side APIs

One of the most critical points to pay attention to is maintenance of connections with side APIs providing data. The data is primarily articles, abstracts and ratings. Since the side APIs used as data sources for the current project are independent programs, there is all probability they will change at some moment. Potential changes can be classified into several groups.

First, outer changes, i.e., those in URL address, parameter names, HTTP headers expected, and the like. This kind of modifications is to be spotted by simply checking the corresponding log. The existing logger records exceptions that will be thrown in case a connection failed or a similar error occurred.

Another type of potential API modifications one should mention is inner structural changes. Due to internal business processes, it usually takes a period of time to get model stable to satisfy different customer needs. In case of API, those optimizations well may result in structural modifications of some API responses. Modifications in API responses, in turn, cause discrepancies in API response structure and the model in current project to parse the response into. Response parsing will not necessary throw an exception, hence the logger used will not record the error. That is why these potential bugs are harder to keep track of.

One (probably the most reasonable) strategy here appears to check from time to time whether an API returns any abstracts at all. In case it does not, one should think about examining the structure of API responses. Another, more resource consuming option is to expand the logger in the way it records also API row answers.

Here is one more important feature to add to make abstract seeking process more effective. This feature has to do with avoiding repetitive search of abstracts. There is a high probability, that most of such searches will not produce a positive result. There is a significant amount of articles in the database for which abstracts have not been found even after several rounds. Every next failed try reduces the chance to ever find abstract for such articles. Taking into account a limited number of calls one can make to almost all available APIs (IEEE, Springer, Google Scholar), avoiding repetitive abstract search is a strategy to make abstract search more effective overall and per call.

There are lots of options to implement this feature, yet probably an optimal one in terms of development resources required and the flexibility of articles extraction from the database is the following:

```
1.  /// <summary>
2.  /// return spicified number of articles that have no abstract
3.  /// </summary>
4.  /// <param name="itemsToSkip"></param>
5.  /// <param name="itemsToTake"></param>
6.  /// <param name="searchMode"></param>
7.  /// <returns></returns>
8.  private static List<Article> GetArticlesWithoutAbstract(
9.      int itemsToSkip, int itemsToTake,
10.     AbstractSearchMode searchMode = AbstractSearchMode.ResourceSaving)
11. {
12.     var articlesWithoutAbstract = new List<Article>();
13.     try
14.     {
15.         using (var context = new ArticleContext())
16.         {
17.             articlesWithoutAbstract = context
18.                 .Articles
19.                 .Include("Abstract")
20.                 .Where(a => a.Abstract == null ||
21.                             a.Abstract.Text == string.Empty)
22.                 //TODO: Where(a =>
23.                 //      searchMode  == AbstractSearchMode.ResourceSaving
24.                 //       ? !a.HasBeenSearhched
25.                 //       : a.HasBeenSearhched || !a.HasBeenSearhched
26.                 .OrderBy(a => a.Id)
27.                 .Skip(itemsToSkip)
28.                 .Take(itemsToTake)
29.                 .ToList();
30.         }
31.     }
32.     catch(Exception ex)
33.     {
34.         //log...
35.     }
36.
37.     return articlesWithoutAbstract;
38. }
```

Program 12. *An extra condition for method GetArticlesWithoutAbstract adds new feature to articles selection to make abstract search optimal*

If implemented, the feature described in Program 12 allows selecting an article extraction mode: either thorough or resource-saving. In the resource-saving mode, articles with the flag “HasBeenSearched” set true will not be included into final selection for abstract search.

Above described aspects of maintenance were only several examples to enhance the overall productivity of the application. There may be lots more depending on further needs.

5.1.4 Collecting Other Types of Academic Information

Besides articles, the DBLP dump file contains metadata on other types of academic information like conference papers, books, internet publications. It would be possible to collect other types of academic data for potential researches with more specific purposes.

5.2 Using the Solution as an API for Side Projects

It is reasonable to assume that this project may be used as a data source for researches associated with side projects. In this case, it is very likely that the data mining software will not be native for the .NET environment. For example, Python programming language is not directly compatible with the languages supported by .NET.

Python is known to be one of the most suitable languages for machine learning and data mining and subsequent results displaying. It presents a number of modules to process large volumes of data with [20]. Here is a couple examples of them: Pandas, Scikit-learn. Thus, it is highly likely that a machine learning algorithm using data current project provides is written on Python language.

Communication between the two nodes, namely the data provider (current project) and the algorithm using it, is a matter of some architectural decision. There are two major directions in this case; either to combine the two modules into one solution or use them separately ensuring communication over some protocol, typically HTTPS.

First approach carries with it certain challenges in terms of integrating Python compiled libraries into a .NET solution. It requires installing an extra interpreter, ensure compatibility of different modules and other challenges that emerge as one tries to integrate two modules that are not natively compatible. These challenges lead us to take a deeper look into the alternative option.

The latter option excludes the need for challenging integration and proves to be more flexible. Communication over HTTPS is fast and stable. It allows side projects to use numerous APIs for as data sources. The second approach also makes it possible for current project to serve as data sources for many independently working data mining projects.

5.3 Completing the Database with Data on Authors

Another attribute to enhance data mining precision could be metadata on authors. Every item in the DBLP dump file contains a list of authors. It is definitely worth considering using the metadata on authors to improve rating predictions. It is reasonable to assume that the set of words and authors uses is smaller and more consistent in compari-

son with publishers' dataset. Because the same authors may be published in several journals, there may be a hypothesis that it is easier to predict natural language patterns or linguistic patterns based on authors rather than on publishers. Intuitively, this hypothesis is not ungrounded and, therefore, might be worth examining.

The data on article authors is included into the database.

6. CONCLUSION

Section 6 outlines briefly major conclusions of current project as well as answer questions ask at the initial stage of this project.

At the initial stage of this project, a number of questions were raised.

The first question asked is connected to the sufficient amount of data needed for data mining. There is not just one correct answer. Depending on a number of factors like research objectives, type of data, available resources, etc. there may be different decisions. As far as current project is concerned, the decision made on the sufficient amount of data (articles abstracts) ensured some degree of accuracy for data mining experiments as well as took into account limited resources (database, web server memory, development costs) available for this work.

Another challenge this project aimed to answer was about the possibility to collect and maintain academic information from different sources with no control on them. The answer is rather positive even though there are certain risks especially regarding smooth maintenance and updating the database.

When it comes to concrete results, it was possible to extract over 1 460 000 article items from the DBLP dump file. The number of article abstracts collected from different sources exceeds 777 000 items. Total number of articles with both JUFO rating and abstract is over 522 000 which makes it possible to conduct a wide variety of researches in the area of data mining and adjacent fields.

Taking into consideration existing JUFO rating mapping algorithm that may be upgraded to map more items remaining unmapped for the moment and side APIs for abstract extraction (like Google Scholar, Scopus or IEEE Xplore) that were not utilized in this project, one may estimate the potential number of completed items as large as 1 000 000. Of course, this will require additional work. Current figure of 522 000 completed items appears to be a sufficient output to begin with.

However, it must be mentioned that when it comes to further maintenance and completing the database with new items, several risks emerge. They are mostly connected to the fact that one has no control on side APIs data structures, connection settings, etc.

And last question of this thesis addressed to the possibility of prediction academic article rating based on its word bag. Although the experiments conducted in this project were rather primitive, the main output of those experiments is that the data collected can

be utilized for researches in natural language processing and other fields related to data mining and machine learning.

To sum up, this work answers all above raised questions with different degree of accuracy. Whereas there is no plain answer on the sufficient amount of data for data mining, it proves to be quite possible to collect academic information from different sources. Also, as this work shows, there are all means to maintain the database as well as add new data although in a semiautomatic mode.

The result obtained after processing a part of generated academic articles is a topic of a discussion.

This project was created in the way it may be extended to fit a wide variety of researches in the field of natural language processing and related areas.

REFERENCES

[1] Dblp: Home, website. Available (accessed on 16.05.2018): https://dblp.uni-trier.de

[2] Dblp: Statistics, publications per year, website. Available (accessed on 16.05.2018): https://dblp.uni-trier.de/statistics/publicationsperyear

[3] Dblp: XML dump file. Available (accessed on 16.05.2018): https://dblp.uni-trier.de/XML

[4] Freeman A., The MVC Pattern. In: Pro ASP.NET MVC 4. Apress, Berkeley, CA, 2012

[5] Vijay P. Mehta, Getting Started with Object-relational Mapping. In: Pro LINQ Object Relational Mapping with C# 2008. Apress, 2008

[6] Microsoft: Developer Network, website. Available (accessed on 31.05.2018): https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx

[7] Microsoft: Developer Network, website. Available (accessed on 31.05.2018): https://msdn.microsoft.com/en-us/library/hh949853(v=vs.113).aspx

[8] Scott Klein, Pro Entity Framework 4.0. Apress. 2010

[9] Freeman A., Networking & WCF. In: Introducing Visual C# 2010. Apress, Berkeley, CA, 2010

[10] Liquid Studio 2017, Available (accessed on 31.05.2018): https://www.liquid-technologies.com/trial-download

[11] Microsoft: Import and Export Data with the SQL Server Import and Export Wizard, website. Available (accessed on 05.06.2018): https://docs.microsoft.com/en-us/sql/integration-services/import-export-data/import-and-export-data-with-the-sql-server-import-and-export-wizard?view=sql-server-2017

[12] Hua Y., Yu H., Zhenwei H., Jianmin Y., Mingming Z., Yanhui F. Combination Method of Rules and Statistics for Abbreviation and Its Full Name Recognition. Advances in Intelligent and Soft Computing, vol 110. Springer, Berlin, Heidelberg

[13] Daniel R. Clark . Introducing Visual Basic and the .NET Framework. In: Beginning Object-Oriented Programming with VB 2005. Apress, 2006

[14] Publication forum: Home, website. Available (accessed on 18.06.2018): http://www.julkaisufoorumi.fi/en

[15] Springer: Home, About us, website. Available (accessed on 20.06.2018): https://www.springer.com/us/about-springer

[16] Semantic scholar: Home, website. Available (accessed on 21.06.2018): https://www.semanticscholar.org/

[17] Institute of Electrical and Electronics Engineers: Home, website. Available (accessed on 21.06.2018): https://www.ieee.org/

[18] Association for computing society: Home, website. Available (accessed on 21.06.2018): https://www.acm.org/

[19] Pérez M.S., Sánchez A., Herrero P., Robles V., Peña J.M. Adapting the Weka Data Mining Toolkit to a Grid Based Environment. In: Szczepaniak P.S., Kacprzyk J., Niewiadomski A. (eds) Advances in Web Intelligence. Lecture Notes in Computer Science, vol 3528. Springer, Berlin, Heidelberg, 2006

[20] Unpingco J. Machine Learning. In: Python for Probability, Statistics, and Machine Learning. Springer, Cham, 2016

[21] Aljawarneh, S., Yassein, M.B. & Aljundi, M. Cluster Comput. An enhanced J48 classification algorithm for the anomaly intrusion detection systems, 2017

[22] Wen W., Hao Z., Cai R. Mining the Discriminative Word Sets for Bag-of-Words Model Based on Distributional Similarity Graph. In: Cai R., Chen K., Hong L., Yang X., Zhang R., Zou L. (eds) Web Technologies and Applications. APWeb 2015. Lecture Notes in Computer Science, vol 9461. Springer, Cham

[23] Random Forests. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning and Data Mining. Springer, Boston, MA, 2017

[24] Devetyarov D., Nouretdinov I. Prediction with Confidence Based on a Random Forest Classifier. In: Papadopoulos H., Andreou A.S., Bramer M. (eds) Artificial Intelligence Applications and Innovations. AIAI 2010. IFIP Advances in Information and Communication Technology, vol 339. Springer, Berlin, Heidelberg, 2010

[25] Elsevier: Home, website. Available (accessed on 27.09.2018): https://www.elsevier.com/

[26] Joshi B. Overview of SOLID Principles and Design Patterns. In: Beginning SOLID Principles and Design Patterns for ASP.NET Developers. Apress, Berkeley, CA, 2016

[27] Weldon W. Nash, Generics. In: Accelerated C# 2008. Apress, 2007

[28] Else, H. How I scraped data from Google Scholar. Nature, https://doi.org/10.1038/d41586-018-04190-5, 2018

[29] The Geeky Geko, WebClient vs HttpClient vs HttpWebRequest, website. Available (accessed on 16.10.2018): http://www.diogonunes.com/blog/webclient-vs-httpclient-vs-httpwebrequest/

[30] Breiman, L. Machine Learning 45: 5. https://doi-org.libproxy.tut.fi/10.1023/A:1010933404324, 2001